下载APP

24 | 监控设计，让一切都有迹可循，尽在掌控

2020-12-18 乔新亮

乔新亮的CTO成长复盘 进入课程 >

讲述：乔新亮

时长 19:47 大小 18.13M

你好，我是乔新亮。

这一讲，我想和你聊聊如何做好监控设计。

你可能会想，为什么要聊监控呢？做监控不是很简单吗？

所有做技术的同学，基本都会根据公司的日志规范，在代码中打印 Log ，以记录告警和报错。许多企业，也会将日志收集分析，以此形成对系统状态的监控。如果条件允许，团队还可以使用各类免费或付费的服务器监控报警服务，多方便啊，这有啥好讲的呢？

其实在我眼里，这些都只是构成了监控的一部分，并非完整的监控体系。要想深刻理解监控的概念，我们首先要学会问自己：为什么要做监控系统？这就像许多工作方法论里强调

的一样，做事先问目的 —— “start with why”。

监控的目标是及时发现系统的问题，并尽可能快地做出相应的动作，让系统一直处于健康的状态。监控，可以拆分为“监”和“控”分别理解，其含义恰好对应着两种主要手段，也就是“**监视**”和“**控制**”（中文真是博大精深）。

比如说，生产环境出现了个 bug，怎样定位问题？这需要做好“监视”；发现问题根源后如何正确响应？这需要做好“控制”。

比较无奈的情况是，研发人员知道系统出了问题，但无法定位问题；更悲催的情况是，研发人员能够定位问题，但无法控制问题，只能眼睁睁地看着故障发生。但无论哪种情况发生了，从结果来看，其实区别都不大。唯一的区别可能是，前者至少让人心存希望，后者则让人感觉整颗心都“冰凉冰凉”的。

说到这里，我不禁有点感慨：监控、监控，意味着既要监视也要控制，没有控制的监视达不到目标，没有监视的控制无法形成行动计划。

从业至今，我曾处理过许多因认知不足，而导致的系统监控问题。下面，我们就选取其中一件比较典型的案例，尝试着一同复盘下。

技术团队的“怪现象”

这个案例发生于某日下班后 —— 大约是在傍晚 20:00 左右，团队发现了一个出现在生产环境的异常。该异常来自一个可以连接众多终端设备的程序，其具体表现为：所在服务器的 CPU 负载突然飙升。但发现异常后，团队既无法将其恢复，也无法定位问题。

发现问题后，共有十几位工程师先后参与了问题分析。很快，两个小时就过去了，监控系统仍然显示 CPU 占用异常，问题还是没有解决。这下，产品负责人着急了，向我求助，并将我拉进了问题调查组的电话会议。

进入会议后，我立刻问了大家一个问题：

“最近线上有什么版本变动吗？都回退了吗？”

团队回答道：“都回退了。”

我几乎没有任何犹豫地说道，不可能，一定没有全部回退。

要知道，计算机是非常可靠的。如果一套代码在一台机器上可以正常、稳定地运行，那么在外部环境没有发生任何变更的情况下，一周、一个月、一年后，它大概率能继续保持正常、稳定地运行。

你可能会想，老乔又开始信口胡说了，难道就没有意外情况吗？

有，我也确实遇见过，但在近 20 年职业生涯里，我只遇见过一次（ZooKeeper 因虚拟机连接数过多而崩溃事件，在“高可用设计”部分，我们曾简单提到）。所以，这种情况几乎是“可遇不可求”的，在大部分时间里都可以直接忽略。

果然，在我的追问下，负责发布的同学说，系统没有回退到上一个版本，而是按研发同学的指挥选择性回退的。

好呀，真相大白，我命令他迅速按公司研发制度回退。

没成想，回退到上一版本后，系统依然不正常。于是，团队继续查看前端、服务端、数据库等各类监控数据，试图分析问题到底出在哪里。

我再一次找到团队，问道，真的都回退了吗？

团队可怜兮兮地回答，真的都回退了！

我的回答是，不可能，鬼才信你们…… 于是，我开始继续不依不饶地追问。

终于，在我的“威逼利诱”下，有位同学一拍脑门，犹豫地说道：“我在数据库里加了条索引，不过这肯定不会导致负载异常……”

还没说完，我就哭笑不得地打断了他的话，谈这些做什么？赶紧回退去。

当这个改动回退后，一切都恢复了正常。

最终，一个本该三分钟内被搞定的生产问题，用了几个小时才解决。幸亏发布窗口是业务低峰期，否则已酿成重大损失。在许多公司里，相关人员几乎一定会受到处罚。

事后复盘时，我问团队，当我询问版本有无全部回退时，为何不及时声明？

那个同学委屈地说，我以为那条索引不可能惹出这么大的问题……

请注意，在这个故事里，当事人员并不是毫无经验的新手；公司并非没有监控系统，相反还做到了可视化、图形化。

既然如此，为何团队还会被一条小小的索引命令阻滞这么久呢？这是一个非常奇怪的现象。而且，在参加 GTLC 全球技术领导力峰会时，我在和一些同学的沟通中也发现：这种现象并非孤例，很多团队都曾出现过类似的情况。

于是，我不得不去思考，IT 团队的系统监控体系，到底出了什么问题？

IT 团队的系统监控体系，到底出了什么问题？

当陷入思考的泥潭时，我们往往需要将思维抽离，尝试站在更高维度探寻问题的本质。此时，我们不妨将这个案例暂且放一放，重新思考一下监控的本质含义。

在文章开头，我们讲过了，**监控是为了让系统一直处于健康状态，具体的手段包括“监视”和“控制”两种。**简单来说，就是当生产环境出现问题时，我们要能知道哪里出了问题，并具备相应的控制手段。

生产环境应急恢复的最大挑战在于根因分析，即找到问题的根本原因，这往往是耗时最久的工作。

但如果没有控制手段，那么即使找到了根因，团队也是无能为力，只能干着急，或者静静地期盼奇迹出现，好像系统在下一分钟就能自动恢复健康。

所以，当生产环境发生异常时，大部分团队会这样组织故障恢复工作：

1. 发现问题后，立即联系各相关系统负责人，以便共同排查问题；

2. 要求大家在一分钟之内回复：自己治下的系统或服务是否健康（这里要将“健康”的定义想清楚，如，响应时间是否增加超过 30% 等）；
3. 进行根因分析，确认导致问题的系统、服务；
4. 完成系统恢复工作。

对于步骤 1、2，其实挑战都不大，真正的挑战在于步骤 3、4。步骤 3 依赖于相关分析人员的专业程度。一般情况下，随着企业业务的复杂度逐渐增加、系统和服务的数量逐渐增加，人员规模也会越来越大，在步骤 3 上花费的时间就会脱离控制。

此外，对于步骤 3 中涉及的不健康组件，我们需要分析：这种不健康状态是原因，还是结果。关于分析方法，其实也比较简单：

1. 首先，我们要确认异常是外因导致，还是内因导致。比如，服务响应慢，既可能是因为外部调用量变大，也可能是因为内部进程繁忙，导致 I/O 、内存、网络资源发生争抢。这步判断相对来说比较耗时间，只有当调查足够充分时，结果才可能浮出水面；
2. 无论是内因还是外因，都要“顺藤摸瓜”，继续进行排查，最后进行恢复。

说到这里，我们不难得出结论：生产应急保障体系的建设，并不像大家想象的那么简单。从系统复杂度到人员专业性，任何不足都会导致问题定位、根因分析花费更多的时间。因此，以上故障恢复流程也存在很大的隐患。

那么，难道生产系统出了问题，我们就只能坐以待毙吗？有没有一个对团队专业度依赖较低的方法呢？答案是，有的。我把它总结为：**流控和版本回退，简单、粗暴、实用**。

流控，就是做好程序的并发流量控制；版本回退，就是在生产环境的发布出现问题时，及时回退到上一个版本。

生产环境出现问题，原因通常只有两个字：变化。常见的“变化”大致有三类：

1. 外部用户请求量增大；
2. 产品发布，一般包括代码发布、配置发布、SQL 脚本发布等；
3. 依赖资源变化，一般是计算、存储、网络基础设施情况变差，比如磁盘存在坏道等。

这样看来，当故障发生时，我们不一定要组织团队进行复杂的根因分析 —— 那是故障恢复后的事儿。相反，只要我们控制住了服务的近期变化，也就等于控制住了故障。思路一变，好像就豁然开朗了。所以， 我们不妨将生产应急恢复方法修改为以下 4 条：

1. 发现问题后，立即联系各相关系统负责人，以便共同排查问题；
2. 要求大家在一分钟之内回复：自己治下的系统或服务是否健康（这里要将“健康”的定义想清楚，如，响应时间是否增加超过 30% 等）；
3. 此处组织两批研发力量，并行工作。第一批解决专业问题，继续跟进问题的定位和调试；第二批负责消灭变化，对有变化的模块进行回退，对于外部请求数量升高的模块启动流控；
4. 恢复系统。

你看，这样就降低了故障恢复对于团队专业性的要求。只要我们保证，对于任何组件，都有以下两种手段同时存在：

1. 流控手段；
2. 发布回退手段。

是不是很简单？在 IT 行业，对于专业团队而言，一般没有太过复杂的工作；如果工作过于复杂，往往是因为负责人不够专业。还是那句话：大道至简。

问题的关键在于，生产环境是不允许查找 bug 的。**在生产环境，研发人员应该寻找并消灭“变化”。从寻找 bug 到寻找变化，是一个非常大的认知转变。**

此时，再回顾文章开头的案例，答案就比较清晰了。团队在执行生产环境故障恢复工作时，接连犯下了三个错误：

错误一：负责发布的同学，没有按规定回退至稳定版本，而是询问开发同学的意见，并以其意见为准；

错误二：相关负责人，因为假设“一条索引不会导致故障”而知情不报，导致系统无法完全回退；

错误三：十几名团队成员没有将精力聚焦在线上业务恢复方面，而是试图在生产环境查找 bug。

三个错误如出一辙，全部来自于技术同学的思维惯性。在大家的认知里，制度就是拿来参考的，对应模块的负责人才是专业的，出了事先找人再问制度规范，或者压根不问制度；在大家的认知里，bug 要用排除法解决，“确定”没问题的模块就要排除在外；在大家的认知里，能一眼发现错误的程序员，才是技术高手，如果动不动就回退，和“网管只会重启”有什么区别？

但是我想说的是，当问题发生时，你的潜意识是要找到 bug ，还是找到变化，对应的结果可能是完全不同的。我们的目标是，**即使找不到 bug，依然可以做好故障恢复**。

这样的思维惯性有多么普遍呢？我们举个例子：你是否曾在下班回家的路上，被老板电话叫回公司，查找线上 bug ？

也许你会觉得这样做很合理，但听完这节课以后，希望你能够意识到这种做法的局限性：线上有 bug ，为什么需要研发立刻回公司？先回退就好了嘛。否则，研发现场写程序，压力也是挺大的。

更别提，一些团队还经常出现喜剧性的一幕：生产环境出现了问题，研发火急火燎地改了个版本，结果系统做不到秒级发布，单是发布就用了半个小时。好不容易上线了，还没来得及高兴，就发现了个新问题……

我们必须不断强调并加深印象：**生产环境永远不允许调试问题，出现问题立刻回退，查问题要去测试环境**。

你可能会想，老乔啊，你这就有点纸上谈兵了，很多大型发布涉及多个系统，怎么可能随便回退呢？

其实这个问题，亚马逊在十几年前就给出解决方案了：**大版本立项，小版本上线** —— 梳理好各模块的依赖关系，将各个系统、各个服务独立发布。当然，这也需要依赖服务版本化和 CI 能力的支持。

以上我们讲的是对企业系统的监控。除此之外，我们也要做好对企业业务的监控。

也就是说，对于企业任何一个未处于理想状态的业务环节，都要进行监控：做好问题的可视化展现、明确管理执行动作，通过数据化管理，不断完善企业的运营效率和运营质量。道理其实还是相通的。

如果能做到监视一切，分析一切，控制一切，“眼”能看见所有，“脑”能洞察一切，“手”能一手遮天，一切业务数字化，一切数据可视化，一切控制可触发，那么，这个企业的数字化水平一定已经很高了。

结语

今天，我们从一个实际案例出发，主要聊了聊关于企业生产环境系统监控的认知和方法：**监控的目的是让系统一直处于健康状态，具体手段则可分为“监视”和“控制”两种；要做好控制，一个重要的方法是做好流控和版本回退。因为在大部分情况下，消除变化就等于消除异常。**

实际上，不单是技术、业务系统需要做好监控，研发管理、团队管理都要做好监控。

关于研发管理，我们在“高可用设计”部分曾提到：风险是经由开发环境、SIT 环境、压测环境、PRE 环境，进入生产环境的。所以我们要做的是严格检查各个环境下的异常。所谓研发管理规范，应该为代码版本进入下一个环境设置准入标准。对于任何异常，都有负责人进行修正。

对于团队管理，我们常常说，组织是结果导向的，但管理工作是过程导向的。关注过程自然就会得到好的结果，只盯着结果往往什么也得不到。对于一个项目、一个产品，乃至于团队的健康度，管理者有没有在关键节点设置监控？有没有针对异常做好控制？其差别是巨大的。

如果你认真学习了前面的内容，可能也会发现，无论是管理还是专业技术，很多关键认知都是相通的。很多时候，是“一理通百理明”。在学习的同时，前后交叉思考，也有助于你在更高的维度上掌握这些知识。

今天，我们就聊到这里，下一讲再见。

提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 23 | 考虑限制，让自己的产品不入险地

下一篇 25 | 异常设计，让错误无处遁形

精选留言 (12)

写留言

yshan
2020-12-18

笔记整理：

1、监控的目标是及时发现系统的问题，并尽可能快地做出相应的动作，让系统一直处于健康的状态。监控，可以拆分为“监”和“控”分别理解，其含义恰好对应着两种主要手段，也就是“监视”和“控制”。...
展开

作者回复: 你好，yshan
总结的真全面，真棒

3

Weehua
2020-12-21

学习了这篇文章，真的感同身受。我们的研发过程中，线上遇到问题几乎大部分都是变更引起的故障，所以第一时间回滚是强行要求，先止血，恢复业务，然后复盘分析，修复解决问题。监控，监是比较容易的，控反而是困难的，怎么让系统在遇到问题的时候快速恢复才是核心。当然，能及时发现问题也是很重要的。

展开

作者回复: 你好，Weehua
简单的办法但最管用，因为符合本质。

1

术子米德
2020-12-20

有个小困惑，生产系统，有人加个索引，这个操作本生没有被监控和日志下来么？
按理说，生产系统的任何改动，都是要被监控和日志，这本该是生产系统监控最基础的工作，否则就像案例中的样子，问了半天都不知道系统已经被修改了，这不是有点搞笑么？

展开

作者回复: 你好，术子米德
你提的这个问题特别对，这其实也是写这篇文章的初衷。对于发布要进行监控，是这样的。这个也有监控，在发布平台也有，但是对应的DBA的忽略掉了它。这就是认知太重要的原因，大家还是要去寻找bug的思路，而不是寻找变化的思路。而大脑很神奇，就彻底忽略了它。
而当时又是下班后的时间，大家也没有去发布平台看，就相信了当事人。
很多问题事后回顾都是这样，哭笑不得，生产的问题大概如此，这也是这么多年看生产问题得出的结论。所以才总结出不是寻找bug，而不是寻找变化的恢复思路，这个认知特别重要，其实今天很多企业，也没有得出这个认知，你想想是不是？

1

流浪地球
2020-12-18

乔老师的本节课程再一次的感同身受，团队内几次最耗时的生产环境问题处理，都是因为版本发布后不及时回退造成的，造成的损失也很惨重。而且最后导致问题的一定是那个第一开始就被“自信”忽略的改动，信誓旦旦的保证这个改动肯定不会出问题，往往被狠狠

打脸，而且这些兄弟的特点都是兜不住了，最后才承认，真是让人苦笑不得。所以，团队内一定要有这个红线，线上问题一定要及时回退所有改动，谁违反开除谁，这是血淋淋...

展开

1

剑八

2020-12-20

先止血
方法步骤：
1.确定哪个系统有问题
2.确定变化原因，代码版本还是流量变高
代码版本问题就回退上个没问题版本...

展开

作者回复: 你好，剑八
从寻找bug，转向寻找变化，消灭变化，消灭问题。然后再寻找bug，彻底解决。

Y024

2020-12-20

其实就“平平无奇”的 log 记录，很多人/产品/项目都做不好，出了问题都只能上重型武器——ide debug。

作者回复: 你好，Y024
是的，其实记好log，本身时监控体系的一部分，如果数据都不对，不准确，后面在寻找问题时会出问题。这也是为什么我们寻找变化，而不是寻找bug的原因，很多信息都没有记录，所以难以定位。

天天向上

2020-12-18

目的 让系统一直处于健康状态

思想 问题源于变化，因此发现变化并及时消灭变化极大概率消灭问题

核心手段 监视和控制 区分内外因后 流控和回退...

展开

作者回复: 你好，天天向上
是的，这是我实际使用的管理办法，包括在现在的公司，这些都是实际在做的事情，效果非常好。

spark
2020-12-18

先分享一个我的经历：以前遇到数据库CPU占比100%，系统不可用30分钟，连续两天都是在早上9:00左右发生。最后排查是一个程序员写了一个定时JOB，把主库的CPU搞到100%。当时我们每天的收入是100万，发生事故后每天10万。
发生以上事故的原因：
不是我负责的(先划清bug引入的责任人)。是我的上级负责的，从管理、系统故障认知和...
展开

作者回复: 你好，spark
真正的负责线上稳定性，业务连续性，就会对这部分深有感触。
一个正确的认知，可以解决掉那些困扰、焦虑、担忧。

worfzyq
2020-12-18

错误一问题存疑，具体可能得看业务场景，如果涉及到c端业务肯定不能草率回滚，另外很对功能上了在下会有pr风险，当然确定这些没问题了确实可以这么做。
展开

作者回复: 你好，worfzyq
谢谢提出问题，我们来讨论一下。
第一，现在有问题，不回退，是业务有问题，甚至是不可用，两者权衡，取舍如何做决定，回退是最优解，因为回退的版本是业务可以正常运行的版本。
第二，发布是在业务低峰期，比如一般公司是在晚上，这时候回退了，查明问题再进行发布心理压力多小
第三，回退带来的问题就是上线失败了，但是业务不受影响；而不回退是业务受影响。如果我们不能保证每次发布一定成功，就必须有应对方案，回退就是一个保底的应对方案。回退+发布在业务低峰期，我的职业生涯中认为就是最优解决方案；回退在项目层级是不好，在业务层级对于业务影响几乎没有，是在发布失败时的最优解。
我们一起讨论，也可以结合一个具体的案来看看，是否还有更优的解，再次感谢提出问题。

海
2020-12-18

大道至简。发布中遇到的问题用回退先解掉。
大牛和高级工程师的区别点更多的是在非技术层面?
项目中的很多问题，最终都变成了非专业知识的问题。

展开

作者回复: 你好，海
产品思维+架构设计+项目管理+专才，可能你说的专业知识指的是专才部分的知识，但是我想说的是，这四部分都是专业知识。
专栏第三部分的内容是专业的复盘，专业是来帮助更高效的解决问题的，我们的存在是让这个世界更美好，因为专业，所以可以更高效的解决问题。
谢谢分享和认可。

Jy
2020-12-18

确实是很好的做法，但我们公司还没能做到这一步，只能逐步改进。
比如业务逻辑出错导致的前端或者后台报错，能监控到但没后续，有一键回滚的能力但回滚了整个这方面的业务功能也就没了，业务不能接受，还是需要花时间来调试补上线。
想问下乔老师有无遇到这种情况呢?

展开

作者回复: 你好，Jy
回退应该是回到上一个稳定的业务版本，这个是在发布前就准备好的。
对于大项目，一个大版本发布的组织，这点比较难，但真的是把自己置于险地。只能在线上解决掉问题，团队的心理压力太大了。
我职业生涯碰到过，所以我再也不想再碰到这种情况，从理论上、概率上就彻底消除掉这种情况，那这个研发过程多轻松。

andy
2020-12-18

看了老师这么多的经验分享，其中包含的知识点非常多，我很困惑一个问题，为什么老师，当然还有很多高手，脑子中能够记住这些知识，而我就觉得有多有麻烦。你肯定也清

楚，技术的知识点挺多的，更不说你现在做的管理了，更是浩如烟海，所以，怎么解决这么多知识点，能够更好的理解呢

展开

作者回复: 你好，andy

我分享的这些内容，都是实际工作中使用到的， 日积月累，学习的内容在实际工作中使用了，会固化到自己的知识体系中，如果再加上分享讨论，更是会理解的深刻，理解的越深刻，就越不会忘记。

就如同打羽毛球，学习击球技巧，学习步法，然后在比赛中使用，最后都忘记了，但在工作中都实际使用起来。

练习武术也如此，最后招式都忘记了， 但是实际是融入搏击中，很多都类似。

2